Development of a Real-Time Stringlines Tool to Visualize Subway Operations and Manage Service at New York City Transit

Boris Suchkov, Mikhail Boguslavsky, and Alla Reddy

In 2013, with real-time train arrival data becoming widely available through its general transit feed specification real-time (GTFS-RT) feed, New York City Transit (NYCT) recognized the potential for new service management tools capable of on-the-fly performance visualization and reporting. This paper describes the process taken by NYCT to develop and evaluate a transit analysis tool that uses information from automatic train supervision countdown clocks, transmitted as GTFS-RT data, to visualize train spacing, movement, and related parameters. The main part of this tool is a new web application that uses stringline (time–distance) charts for monitoring operations in the system as they happen. Operational problems (e.g., delays, train bunching, or gaps in service) can be identified more easily on stringline charts than on the traditional model board display to help managers and operating personnel deliver continuous service performance improvements. The application was built in-house with the use of existing data resources (the GTFS-RT feed of predicted train arrivals) and open-source tools. In-house development minimized cost and allowed for maximum flexibility to add features in the future. The development team successfully employed a methodology of iterative development to incrementally review, add features, and test the application without having to go through very long design and requirements cycles. As more transit agencies release GTFS-RT data to the public, they may see opportunities to take a similar approach to develop new tools for internal use quickly and with little additional cost.

The New York City subway carries more than 5 million passengers on an average weekday, serving 468 stations. Historically, the subway system was built and operated as three distinct systems: the Interborough Rapid Transit Company (IRT), the Brooklyn–Manhattan Transit Corporation, and the Independent Subway System. Several years ago, the lines corresponding to the original IRT (Lines 1 to 6 and the Grand Central Shuttle) were upgraded to a system of centralized monitoring and control called automatic train supervision, A division (ATS-A). This system made it possible to digitally report train movements both internally to the Rail Control Center (RCC) and externally to customers by countdown clocks at stations. In late 2012, New York City Transit (NYCT) made countdown clock data available on the Internet for use by web-enabled devices through the general transit feed specification real-time (GTFS-RT) feed. Communications-based train control (CBTC) offers similar capabilities on the L line. Efforts to provide similar functionality are underway for the rest of the subway system.

This paper describes the process taken by NYCT to develop and evaluate a web application that uses GTFS-RT data to present near real-time train movements in stringline (time–distance chart) format. It is the latest project in a line of initiatives focused on building an automatic performance-reporting system.

B. Suchkov, Cubicle A17.111; M. Boguslavsky, Cubicle A17.110; and A. Reddy, Office A17.92, System Data and Research, Operations Planning, New York City Transit, 2 Broadway, New York, NY 10004-2208. Corresponding author: A. Reddy, Alla.Reddy@nyct.com.

Transportation Research Record: Journal of the Transportation Research Board, No. 2538, Transportation Research Board, Washington, D.C., 2015, pp. 19–29.
DOI: 10.3141/2538-03

BACKGROUND

Evolution of Performance Reporting at NYCT

Historically NYCT had evaluated service performance for the entire subway system on the basis of data collected manually, according to a random sampling plan, at time point stations (*1*). Four years ago, access to ATS-A data enabled NYCT to expand on its reporting and performance indicator framework. From that time on, key performance indicators such as wait assessment (a measure of headway adherence that is NYCT's primary subway service performance metric) could be calculated using 100% data for Lines 1 to 6, instead of a sample, and data became available for every station on those lines. The availability of automatic data and custom software allowed for the indicators to be algorithmically evaluated and reported on a daily basis (*2*). Two years later, a web-based reporting system built on a custom-configured business intelligence reporting suite became available for a pilot project. Its initial success was proved in processing and reporting data from BusTime and related NYCT systems (*3*).

In 2013, with real-time train arrival data becoming widely available for Lines 1 to 6 (*4*), NYCT recognized the potential for new service management tools capable of on-the-fly performance visualization and reporting. However, no such capability existed in the reporting system. This project builds on previous research and development efforts to build a core transit analysis tool that uses GTFS-RT data from countdown clocks to visualize train spacing, movement, and related parameters. The main part of this tool is a new web application that uses stringline charts for monitoring operations in the system as they happen and helping deliver continuous service performance improvements.

Rail Control Center

NYCT subway operations are largely centralized at its RCC. While for the B division (the lettered lines, except for the L) and the Line 7 control over the interlockings and train position recording remains

with the tower dispatchers, service on Lines 1 to 6, Grand Central Shuttle, and L line (more generally, the ATS lines) is controlled by dispatchers at the RCC. Trains on Lines 1 to 6 and the Grand Central Shuttle are operated manually, with train identification (ID), train tracking, and interlocking control automatically provided by the ATS-A system. Automatic train operation is available for the L line (the CBTC ATS system). For all lines, the RCC provides incident management and announcement services.

At the RCC, ATS system dispatchers have a live view of the position and status of every train by their consoles as well as large-screen model boards. These views closely mimic the traditional track-circuit view of a tower operator's control board and are not to scale. Consoles also allow for the manipulation of train service patterns and information (such as setting train IDs and assigning schedules). A limited number of canned reports are available for statistical purposes. A subset of the real-time data is pumped to reporting servers, the countdown clocks at stations, and the public announcement system.

Although console dispatchers have access to the location and schedule of every train, they are also limited in their view of the subway system as a whole and in how trains move relative to each other. The ATS-A and CBTC systems, respectively, were built by outside contractors with limited knowledge of NYCT's internal reporting mechanisms. For example, it is apparent that the ATS-A headway regularity measure is not calculated by the same process as the official wait assessment measure. Because NYCT runs a very frequent service, especially at peak hours, it has been recognized that headway regularity is an important customer-oriented indicator (*2*). Dispatchers could use a new way to identify too-large or too-small headways at a glance, as well as to cross-reference train location data with other proprietary NYCT information sources. These data include data about flagging (stringent safety measures such as lower speed limits required by maintenance and repair work done on the railroad while it is in service) and data about incidents, which can prompt emergency changes in service.

Stringline Charts

Stringline charts are time–distance plots most often used to illustrate transit vehicle movements. Showing time on one axis and stations (spaced proportionally to their actual distances) on the other, each vehicle appears as a string moving diagonally across the plot. Representations of stringlines have existed since at least the 19th century; some were made using actual string pinned on sheets of paper. Perhaps the most famous example, Marey's graphical train schedule (1885), is illustrated in Tufte (*5*). The main benefit of a stringline chart is that it allows one to see both the broader picture of a transit operation as well as its details. At a high level, the plot may show a day's worth of train movements. With a closer look, one can make a number of conclusions related to service patterns. Because the slope of each line segment is proportional to speed, relatively faster or slower services can be readily recognized (with horizontal sections of each line representing dwell time at stations). Train spacing, passing points, and scheduled connections can all be analyzed in ways that would be difficult or impossible to do with just tables of numbers.

In recent years, transit agencies around the world have started using computerized versions of stringline charts to observe and manage transit service. This use is a part of a larger trend toward using more quantitative data (especially automatic vehicle location data) in transit performance analysis and developing performance measures to evaluate transit service (*6–8*). Visualizing a schedule using a stringline chart can help pinpoint irregularities and bangs (conflicts where multiple trains are scheduled to be at the same location at the same time) and provide insights for improving service. Visualizing actual data can help management pinpoint operational problems, such as bunching or gaps in service. Stringlines have come to be seen as important tools in the analysis of incidents, delays, and general service patterns.

General Transit Feed Specification

The GTFS was developed to be a common format for public transportation schedules and associated geographic data (*9*), and it has since been widely adopted (*10*). The data standard has enabled analysis of operations across multiple agencies (*10*) and been used to generate transit networks for transportation modeling (*11–13*). The capability to provide real-time vehicle movement information (compared to static schedules) was a subsequent extension of GTFS (*9*). In addition to NYCT, several agencies have already released train arrival prediction data to the public using the GTFS-RT format, including Bay Area Rapid Transit (*14*), Massachusetts Bay Transportation Authority (*15*), and TriMet (*16*).

PROJECT MOTIVATION

On the basis of the availability of data, stringlines have traditionally been evaluated on a post hoc basis. They were reviewed in light of other available data to interpret and corroborate decisions or plans made at the time of service. However, post hoc analysis does not allow for service management as it is happening; it is intended for learning lessons to be applied later, not fixing (or avoiding) problems immediately. Using stringlines as an operational tool would allow for continuous feedback on service actions taken, albeit without some of the extra information available with retrospective analyses.

There was no readily available solution for a stringline chart tool that would provide real-time feedback to NYCT operations staff. The ATS systems as originally built had not made real-time train movement data available in a form that could support stringlines, and it would have been a slow and expensive process, requiring work from the contractor, to reconfigure the systems for that specific purpose. Development of the GTFS-RT feed was a separate, public-facing project that leveraged existing data processing for in-station countdown clocks. Providing the GTFS-RT feed to the public had the added benefit of making it possible to experiment internally with real-time data at no marginal cost to the agency.

A solution for correcting the informational shortcomings at the RCC using automated data was first fully articulated by Frumin as an internal research paper documenting work with RCC dispatchers and management (*6*). It consists of the following main elements:

- Develop real-time stringlines. Data feeds derived from the ATS systems can power a stringline chart tool. This tool can show headway regularity, on-time performance, and station track occupancy time, informing dispatcher decisions about speeding up or holding back service and providing additional information for incident management.
- Integrate incident and flagging data. Root cause analysis of delays would almost certainly include analysis of incident data and data about construction and inspection activities. Such data can also be visualized on top of the stringlines.

- Build a web-based solution. As a relatively simple product, the stringline chart tool can be built using web standards. It would then be possible to run it on any intranet-connected computer without installing custom software.
- Use experimentation and rapid prototyping. Software development is often iterative. Especially when testing out new ideas, it is easiest to iterate over many (imperfect) versions of a product, without worrying, at the early stages, about usability, quality control, or deployment. It is more important to figure out the core product; it can always be polished later.

By combining new web capabilities, easier access to large amounts of automated subway data, and research into stringlines, it became possible to build a product that uses GTFS-RT data to visualize service delays, bunching, and gaps, providing live feedback to ATS dispatchers. It was also an opportunity to test new methods for rapidly and iteratively building transit service management applications using open-source technology. This approach had the potential to open doors, figuratively and literally, and enable closer cooperation between NYCT departments.

METHODOLOGY

In 2012, the Operations Planning unit of NYCT selected a new, web-based system for reporting performance indicators. As part of the deployment, several reports described in Levine et al. (*2*) were ported over to the new system. In early 2013, it was decided to build a report that includes a stringline chart (see Figure 1). This initial attempt was a way to visualize data from existing next-day reporting. It included color coding of stringlines to represent on-time performance status (early, on time, late, or unknown or extra service) and color coding of rectangles placed to represent scheduled headways (colored according to the wait assessment standard). While useful, this report proved to be unsatisfactory in a number of ways. It tested the limits of the reporting tool charting engine and was unreliable. Only a maximum of two subway lines could be represented at one time. It only allowed for next-day analysis.

The next iteration followed when the GTFS-RT countdown clock data feed became available for Lines 1 to 6 and the Grand Central Shuttle. Designed for public use and available through the agency website, this feed did not have the complete data already available internally on a next-day basis, but it proved to be a major milestone: all real-time train movement data subsequently referred to in this paper is based on the GTFS-RT feed. Using this feed, an algorithm was developed that essentially deconstructed countdown clock predictions into station arrival data. Following initial algorithm design, and in parallel with its development, a web application was developed using the D3.js JavaScript visualization library to graphically show the train movements described by the station arrival data. The web application is hosted on a virtual server set up by NYCT's technology department, where it is available on any computer with access to the NYCT intranet. Back-end data processing takes place on the same server in Python, a popular open-source programming language. The back-end process outputs a new set of tab-separated value files containing train movement information approximately every 30 s; these are read by the front-end web application, which automatically refreshes the data it displays. Both the back-end and front-end parts of the project adhered to the lightweight, iterative software development process typical of start-up companies and other small software development concerns.

Back End: Inferring Actual Train Movements from Predictions

NYCT's GTFS-RT feed was released in late 2012, in conjunction with the MTA Subway Time app for the web and Apple mobile devices (*17*). The data feed was designed to enable developers to create applications on other platforms that replicate the functionality of in-station countdown clocks, specifically to determine how many minutes away the next several trains are from a given station. The feed comprises several messages, the main one of which is TripUpdate (*18*). Other messages, namely VehiclePosition and Alert, enable detection of trains that are not moving or delayed; this information is important for countdown functionality, but only the TripUpdate message was used for the stringline chart project.

The main challenge in using the GTFS-RT feed for performance visualization and analysis was to infer actual train movements from a series of predictions. The feed transmits a new TripUpdate message every 30 s, and each message has a StopTimeUpdate message within it, which is essentially a snapshot of future stops and arrival times for trains that are en route or scheduled to depart within approximately the next hour. Stop times from the past are not included; after a train has arrived at a stop, that stop rolls off the predictions list and is no longer available in information about that trip in future messages. Therefore the last forecast generated for a train at a stop is the closest time to the train's actual arrival at that stop.

Finding the Last Prediction

The first algorithm for finding the last forecast followed the existing processing of next-day train movement data, where each trip is represented by multiple rows in a relational database, with each row corresponding to one station stop for one trip. Trips are identified uniquely within each day by a train ID, and for each train ID–station combination, two times were stored: a predicted arrival time and the time stamp of the message in which the prediction was made. The algorithm parsed every new GTFS-RT message into the one-row-per-train ID–station format, and checked whether each train ID–station combination in the message already existed in the database. If it did not, it was added. If it did, the predicted arrival time and the time stamp were updated with the corresponding values from the new message, provided that the new time stamp was later than the existing one, which should be the case if the new message had not been seen by the algorithm before. Table 1 shows an example of the update process for portions of two trips, with the identified last predictions shaded in light gray, and the latest time stamps shaded in dark gray.

The number of comparisons required in this algorithm was large: at a rough approximation, each new message had [6 subway lines × 10 trains forecast per line × 2 directions × 30 stations per train] rows to compare with all of the existing rows collected from previous messages, the number of which grew slowly throughout the course of a day as trips were completed and new trips were added. The processing of each new message took several minutes, and because this was longer than the 30-s interval between messages, a type of buffering was required in the implementation to prevent missing new messages and losing data as the process was running. Still, this algorithm processed the data in near real time, lagging the current time by 10 to 15 min. This result was a considerable speed-up relative to existing next-day reporting and enabled the implementation of the real-time stringline application.

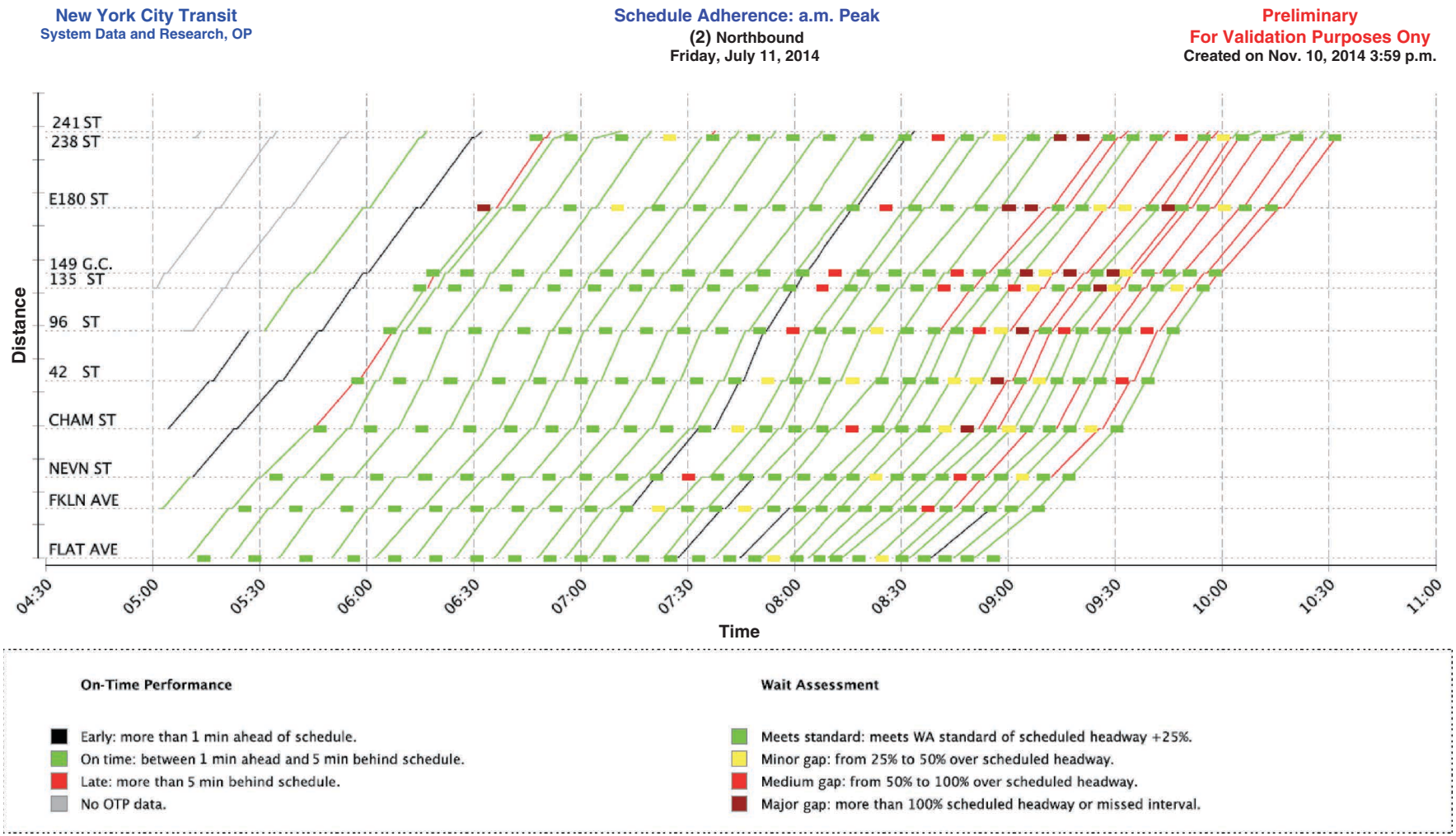

FIGURE 1 Next-day stringline chart with performance indicator data displayed. This early version of a stringline chart tool is a visualization of a next-day report showing the on-time performance (OTP) and wait assessment (WA) indicators. It is run after entering the line, direction, date, and time period selections. The colors of the stringlines correspond to the OTP status of each train at each timepoint. The rectangles correspond to scheduled headways and are colored according to how well actual service met the WA standard.

TABLE 1 Update Process from One GTFS-RT Message to Another

Train ID	Station	Forecast Arrival	Time Stamp	Train ID	Station	Forecast Arrival	Time Stamp
10:03:09 Message				10:05:09 Message			
01 0945 SFY/242	CHAM ST	9:48:33	9:50:39	01 0945 SFY/242	CHAM ST	9:48:33	9:50:39
01 0945 SFY/242	FKLN ST	9:51:45	9:51:39	01 0945 SFY/242	FKLN ST	9:51:45	9:51:39
01 0945 SFY/242	CANL ST	9:52:46	9:52:39	01 0945 SFY/242	CANL ST	9:52:46	9:52:39
01 0945 SFY/242	HSTN ST	9:54:07	9:54:09	01 0945 SFY/242	HSTN ST	9:54:07	9:54:09
01 0945 SFY/242	CHST SQ	9:55:30	9:55:39	01 0945 SFY/242	CHST SQ	9:55:30	9:55:39
01 0945 SFY/242	14 ST	9:56:45	9:56:39	01 0945 SFY/242	14 ST	9:56:45	9:56:39
01 0945 SFY/242	18 ST	9:57:54	9:57:39	01 0945 SFY/242	18 ST	9:57:54	9:57:39
01 0945 SFY/242	23 ST	9:59:15	9:59:09	01 0945 SFY/242	23 ST	9:59:15	9:59:09
01 0945 SFY/242	28 ST	10:00:06	10:00:09	01 0945 SFY/242	28 ST	10:00:06	10:00:09
01 0945 SFY/242	34 ST	10:01:22	10:01:39	01 0945 SFY/242	34 ST	10:01:22	10:01:39
01 0945 SFY/242	42 ST	10:02:58	10:03:09	01 0945 SFY/242	42 ST	10:02:58	10:03:09
01 0945 SFY/242	50-BWAY	10:05:33	10:03:09	01 0945 SFY/242	50-BWAY	10:05:29	10:05:09
01 0945 SFY/242	59 ST	10:07:33	10:03:09	01 0945 SFY/242	59 ST	10:07:29	10:05:09
01 0945 SFY/242	66 ST	10:09:03	10:03:09	01 0945 SFY/242	66 ST	10:08:59	10:05:09
01 0945 SFY/242	72 ST	10:10:03	10:03:09	01 0945 SFY/242	72 ST	10:09:59	10:05:09
		…				…	
01 0952 SFY/242	CHAM ST	9:57:51	9:58:09	01 0952 SFY/242	CHAM ST	9:57:51	9:58:09
01 0952 SFY/242	FKLN ST	9:59:14	9:59:09	01 0952 SFY/242	FKLN ST	9:59:14	9:59:09
01 0952 SFY/242	CANL ST	10:00:23	10:00:09	01 0952 SFY/242	CANL ST	10:00:23	10:00:09
01 0952 SFY/242	HSTN ST	10:01:39	10:01:39	01 0952 SFY/242	HSTN ST	10:01:39	10:01:39
01 0952 SFY/242	CHST SQ	10:03:05	10:03:09	01 0952 SFY/242	CHST SQ	10:03:05	10:03:09
01 0952 SFY/242	14 ST	10:04:46	10:03:09	01 0952 SFY/242	14 ST	10:04:20	10:04:39
01 0952 SFY/242	18 ST	10:05:46	10:03:09	01 0952 SFY/242	18 ST	10:05:44	10:05:09
01 0952 SFY/242	23 ST	10:06:46	10:03:09	01 0952 SFY/242	23 ST	10:06:44	10:05:09
01 0952 SFY/242	28 ST	10:07:46	10:03:09	01 0952 SFY/242	28 ST	10:07:44	10:05:09
01 0952 SFY/242	34 ST	10:08:46	10:03:09	01 0952 SFY/242	34 ST	10:08:44	10:05:09
01 0952 SFY/242	42 ST	10:10:16	10:03:09	01 0952 SFY/242	42 ST	10:10:14	10:05:09
01 0952 SFY/242	50-BWAY	10:12:16	10:03:09	01 0952 SFY/242	50-BWAY	10:12:14	10:05:09
01 0952 SFY/242	59 ST	10:14:16	10:03:09	01 0952 SFY/242	59 ST	10:14:14	10:05:09
01 0952 SFY/242	66 ST	10:15:46	10:03:09	01 0952 SFY/242	66 ST	10:15:44	10:05:09
01 0952 SFY/242	72 ST	10:16:46	10:03:09	01 0952 SFY/242	72 ST	10:16:44	10:05:09

NOTE: This table provides an example of how the last forecast arrival times are identified for two trips. The last forecasts observed are shaded in light gray; the time stamp of the latest message is shaded in dark gray. (Although the message frequency is 30 s, messages 2 min apart are shown here to illustrate a larger change.) In the 10:03:09 message, the forecast arrival time for the 01 0945 SFY/242 train at 42nd Street is 10:02:58, and its forecast arrival time at 50th Street is 10:05:33. The 10:05:09 message did not have an updated forecast for this train at 42nd Street (because the stop was removed from subsequent predictions after the train arrived there), so the 10:02:58 time was retained from the earlier message. The prediction at 50th Street does change as the train gets closer to the station, so it is updated from 10:05:33 to 10:05:29.

A refinement to the first algorithm was able to improve the processing time significantly by using a different data structure to reduce the number of comparisons. The second algorithm bypassed relational databases and the one-row-per-train ID–station format they required. As implemented in Python using nested dictionaries, the second algorithm stuck close to the original GTFS format of the messages, using each train ID as a key to another dictionary that contained the latest predicted arrival time and corresponding time stamp for each stop for that train ID only. Instead of comparing every train ID–station combination in a new message to every existing train ID–station combination as in the first algorithm, the second algorithm first compared each train ID to every existing train ID, and then updated the times at each station only within a single train ID. Effectively this replaced one big comparison with two much smaller and faster comparisons. Pseudocode for this algorithm is presented in Figure 2.

Additionally, because of how dictionaries are implemented internally in Python, the time it takes to test membership in them does not grow (on average) as the size of the dictionary grows (*19*). As a result, it takes effectively the same time to test whether a train ID from a new message is in the accumulated dictionary during the morning, when the accumulated dictionary is relatively small, and during the evening, when the accumulated dictionary is several times larger. This means that performance would not be improved by adding logic to the algorithm that would identify completed trips and exclude them from the comparison, or by dividing accumulated data for the entire day into smaller blocks. The second algorithm can run in random-access memory without writing to disk, but the implementation does write the accumulated dictionary to disk after every message to prevent the loss of the day's data up to that point in the event of a hardware crash or reset.

```
all_day_dictionary ← {}   // Initialize empty dictionary once, at start of the day

for each new message
    // Collect the data into a nested dictionary where the keys of the outer dictionary are the
    // train IDs, and each train ID key points to an inner dictionary of station IDs for that trip.
    // In the inner dictionaries, each station ID key points to a (forecast time, timestamp) pair:
    new_message_dictionary[train_id] → station_dictionary
    station_dictionary[station_id] → (forecast_time, timestamp)
    // and
    new_message_dictionary[train_id][station_id] → (forecast_time, timestamp)

    // The loop below executes the fast comparison of data from a new message to the data
    // accumulated so far that day
    for each train_id in new_message_dictionary
        if train_id in all_day_dictionary
            for each station in new_message_dictionary[train_id]
                all_day_dictionary[train_id][station_id] ← new_message_dictionary[train_id][station_id]
                // if train ID was seen before, replace earlier forecast times with those from new message
        else
            all_day_dictionary[train_id] ← new_message_dictionary[train_id]
            // if the train ID is new, add it along with all of its forecast times
```

FIGURE 2 Pseudocode for algorithm to find the last prediction.

The processing time per message is several seconds, compared with several minutes for the first algorithm. This allows the second algorithm to easily keep up with messages published at 30-s intervals, and time is available for additional data processing that may be required.

Distinguishing Forecast and Actual Train Movements

Testing revealed that the process of saving data from multiple messages (in contrast to working with only the latest message, as the feed is designed to be used) created the need for a data-cleaning step. Specifically, the exception case of a trip that had been forecast at one time but then did not actually run looked the same as typical trips that did run. Such exceptions needed to be removed, and the logic for doing this is relatively simple.

As noted above, each trip has two time values associated with every stop: the last prediction itself and the time stamp of the message in which the last prediction was made. A typical trip will have a different time stamp for the last prediction at each stop, because the forecast time for a given stop will be updated in every message until the train arrives at that stop. (As a consequence of this behavior, the last forecast values are usually very close in time to their corresponding time stamps.) In contrast, a trip that had once been forecast but did not actually run will have the same time stamp for each stop, because those predictions were all made in the same message and then not updated again.

On the basis of this logic, the cleaning step counts the number of times each time stamp appears for the stops within each trip, and if the count is greater than one for any time stamp, the predictions made at that time stamp are flagged as not actual times. This reasoning assumes that a train cannot stop at two different stations within the time interval between messages, which in this case is 30 s. The time stamp of the newest message available is the "current time" for the purposes of the algorithm (generally lagging the true current time by 30 to 60 s). If the predictions flagged as not actual are in the future relative to the newest time stamp, they are identified as current forecasts. These are the unfilled points on the right of the stringline plot in Figure 3; the vertical boundary between these points and the filled points to the left, which represent actual arrival times, is the newest time stamp. Predicted times are also shown in Table 1; they are the arrival times associated with the blocks of identical time stamps.

If predictions are flagged as not actual and they are in the past relative to the newest time stamp, they are removed. This logic cleans out entire trips that had been forecast but not run. It also removes individual stations from trips with en route service changes, where a train had been initially forecast to stop but did not actually stop, either because the train skipped the stations to even out spacing or make up time, because it was rerouted to another track in response to an incident, or because it was short-turned or discharged. The results of this approach were compared extensively to the data on actual train movements reported internally on a next-day basis, which confirmed that the above logic for distinguishing forecast train movements from actual train movements generated correct results.

Front End: Stringlines Web Application

Determining the Platform: D3.js

Shortly after the initial version of the last-prediction algorithm became operational, work began on a web application that could graphically show the train movements and allow for some interactivity and visual analysis. The basic design concept was familiar: time–distance charts similar to those developed for next-day reporting. A search was done first for web-based charting tools, but they proved to be unsatisfactory. Line graph tools are generally optimized to show many points, but only a few chart series; in other words, they have poor performance when showing many disconnected line segments, and stringlines are exactly that: up to several hundred disconnected line series at a time. The search was thus expanded to various general-purpose data visualization packages, with a focus on client-side web technologies.

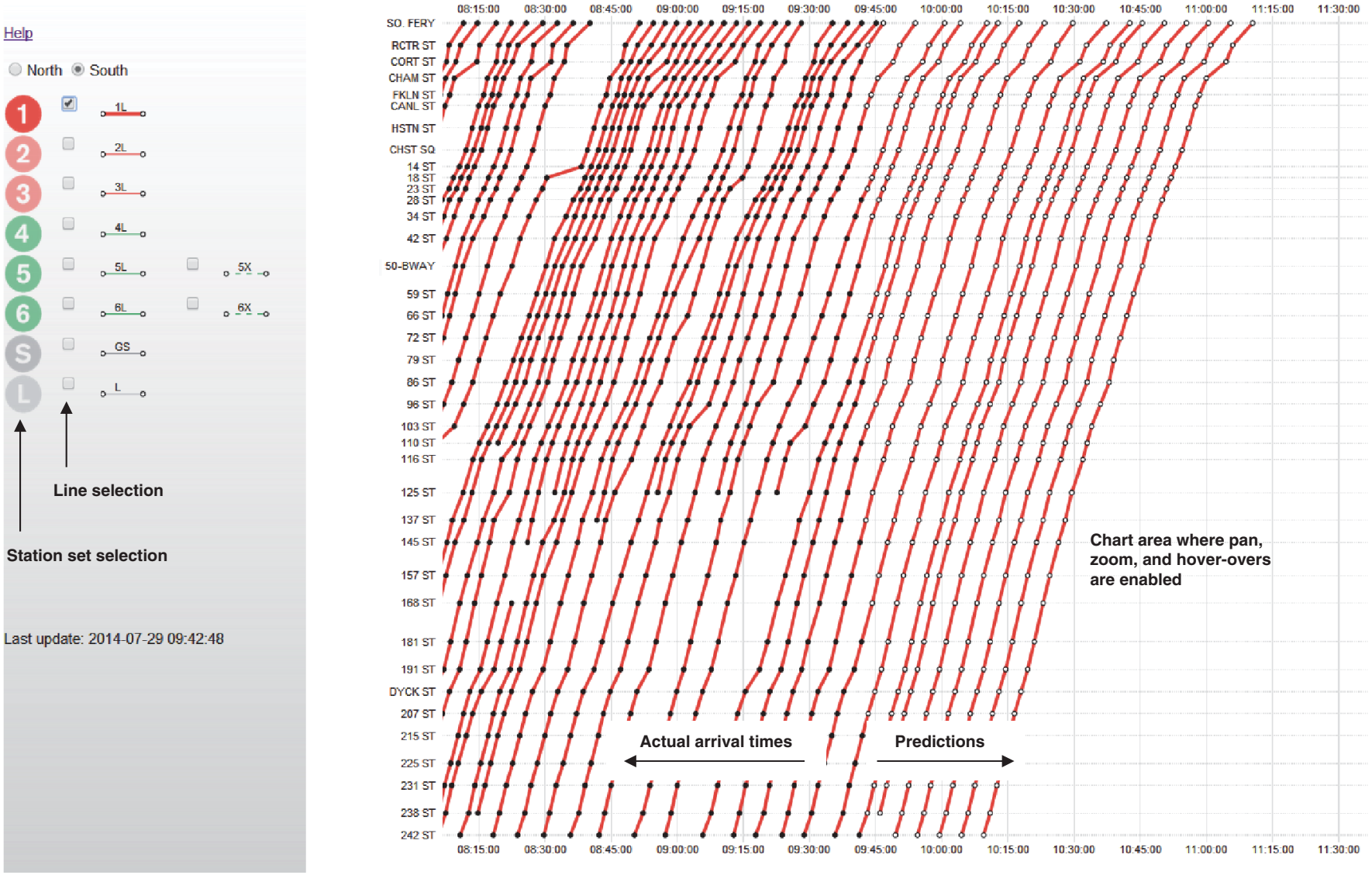

FIGURE 3 Real-time stringlines web application interface.

One that stood out was D3.js, developed and maintained by Mike Bostock (*20*). As its website states, "D3.js is a JavaScript library for manipulating documents based on data." As a library that uses capabilities that come standard in any modern web browser, such as HTML, Scalable Vector Graphics, and Cascading Style Sheets, it is adaptable to many different visualization scenarios. It is fast, does not require any proprietary plug-ins, and provides enormous flexibility in presentation. The central idea behind it involves binding arbitrary sets of data to arbitrary sets of HTML elements (which do not even have to be visual elements). There is therefore no performance issue with line series or any other chart-specific concept. The library is a part of a large and growing ecosystem of D3.js developers, user groups, and example visualizations. The web version of Marey's Trains created in D3.js was used as a proof of concept for this project (*21*).

User Interface Development

Once the decision was made as to which visualization library to use, it became a matter of creating a functional initial version of the stringlines web tool. First, the tool had to show station stops (as dots) and the lines connecting them. Unlike static representations of stringlines, the tool needed to differentiate between stops already made and forecast stops and to offer an easy way to manipulate stringline charts by selecting stations or lines, scrolling or panning, and zooming.

Because the train movement data originate as forecast data, at any point in time it is possible to show both historical data and forecast data. Historical (actual) arrival times are shown as filled points, while forecasts are shown as unfilled points. Separating the two is a vertical line representing the current time (or, more precisely, the time of the last update from the web server).

Initially, the tool appeared as a static web page with a left-side toolbar. The toolbar allowed for selections that are station-centric and selections that are train-centric (see Figure 3). Selecting a station set (for Lines 1 to 6 or the Grand Central Shuttle) by clicking on one of the line icons causes the vertical axis to switch to showing those stations, laid out proportionally to their actual track distances. Only one station set can be selected at a time (since there is only one vertical axis). Selecting a train set corresponding to one or more lines by checking the desired box or boxes shows the train services as viewed from the selected stations. For example, if the station set for Line 5 is selected and the train sets for Lines 2, 5, and 6 are selected, all trains that make stops at Line 5 stations would be shown. Stops for Line 2 in Brooklyn and the Bronx (where Lines 2 and 5 share stations) would be visible, while Manhattan's part of the Line 2 train stringline would be just a straight diagonal line with no points indicating stops, because in Manhattan, Line 2 runs separately from Line 5. Line 6 would show stops in Manhattan, because that is the section it shares with Line 5. If there is no overlap, nothing would be shown. For example, selecting Line 1 station set but Line 6 train set would show only a blank graph, because Line 1 shares no stations with Line 6.

Later project iterations added a number of convenience features that allowed for easy viewing of large amounts of stringline data:

- Zoom. A feature was added for zooming on the time dimension (station spacing on the vertical axis would remain fixed).
- Panning. A panning feature allows for dragging the chart to show earlier or later times of the day without having to scroll.
- Hover-over bubbles. When hovering the mouse pointer over a dot representing a station stop on each stringline, a pop-up information bubble shows the train ID and arrival time at the station.

Lightweight, Iterative Development Process

At a time of severe budgetary constraints at many public transit agencies, it is often difficult or impossible to pursue a large-scale information technology effort, especially one that is experimental in nature. However, combining the latest software development practices and using existing data resources and open-source software, it is increasingly possible to quickly and easily deploy pilot technologies without the need for a traditional systems engineering process. Using approaches more common at start-ups and Internet companies can lead to quick results that can serve as a proof of concept for further efforts.

Being mindful of the RCC information access recommendations described earlier, the stringlines application was built in-house with existing data resources (the GTFS-RT feed) and open-source and open-access tools, including a Linux web server, Python, HTML, JavaScript, and the D3.js library. In-house development minimized cost and allows for maximum flexibility to add features in the future. The development team emphasized a philosophy of iterative development to incrementally review, add features, and test the application without having to go through very long design and requirements cycles. Early on in the project, status updates to management were provided as often as biweekly.

Without a contractor or intermediaries, the conversation between the client and the development team is ongoing and features can be requested and developed on demand. By keeping development in Operations Planning, the conversation between the client and the developer has gone from one in which the client reviews requirements (with the hope that the final product matches both the letter and the spirit of the requirements) to one in which the client reviews the actual product at the time and can immediately comment on everything from usability to data sources as actually developed. While only shown to work with smaller projects, this "best effort" model appears to be effective in driving technical innovation and setting the stage for more formally structured work as needed.

Embracing Standards Allows for Easy Upgrades and New Features

Combining a lightweight, iterative development process with standards-based technology has the additional benefit of easy expansions and modifications. As mentioned previously, although both Lines 1 to 6 and the Grand Central Shuttle on the IRT and the L line have ATS, the underlying technology is different between the two. The GTFS-RT format, however, presents both identically. While the stringlines application was originally built for Lines 1 to 6, adding the L line when its GTFS-RT feed came online (*22*) proved to be trivial; both the back-end algorithms and the front-end web application already supported everything the L line data feed provides, and only cosmetic changes needed to be made to the file structure and HTML documents.

Adding data from other administrative systems at the RCC has proved to be only a slightly larger challenge. For example, connecting to the flagging database required certain data replication and cleaning steps. After that, data integration followed a familiar process. Flagging involves work in both time (with specific starting and ending times) and space (a flagging event typically covers a length of track between two or more stations). Therefore, flagging events can be represented as a set of rectangles on top of a time–distance chart. This process can again be done using the universal visualization capability of D3.js (Figure 4*d*).

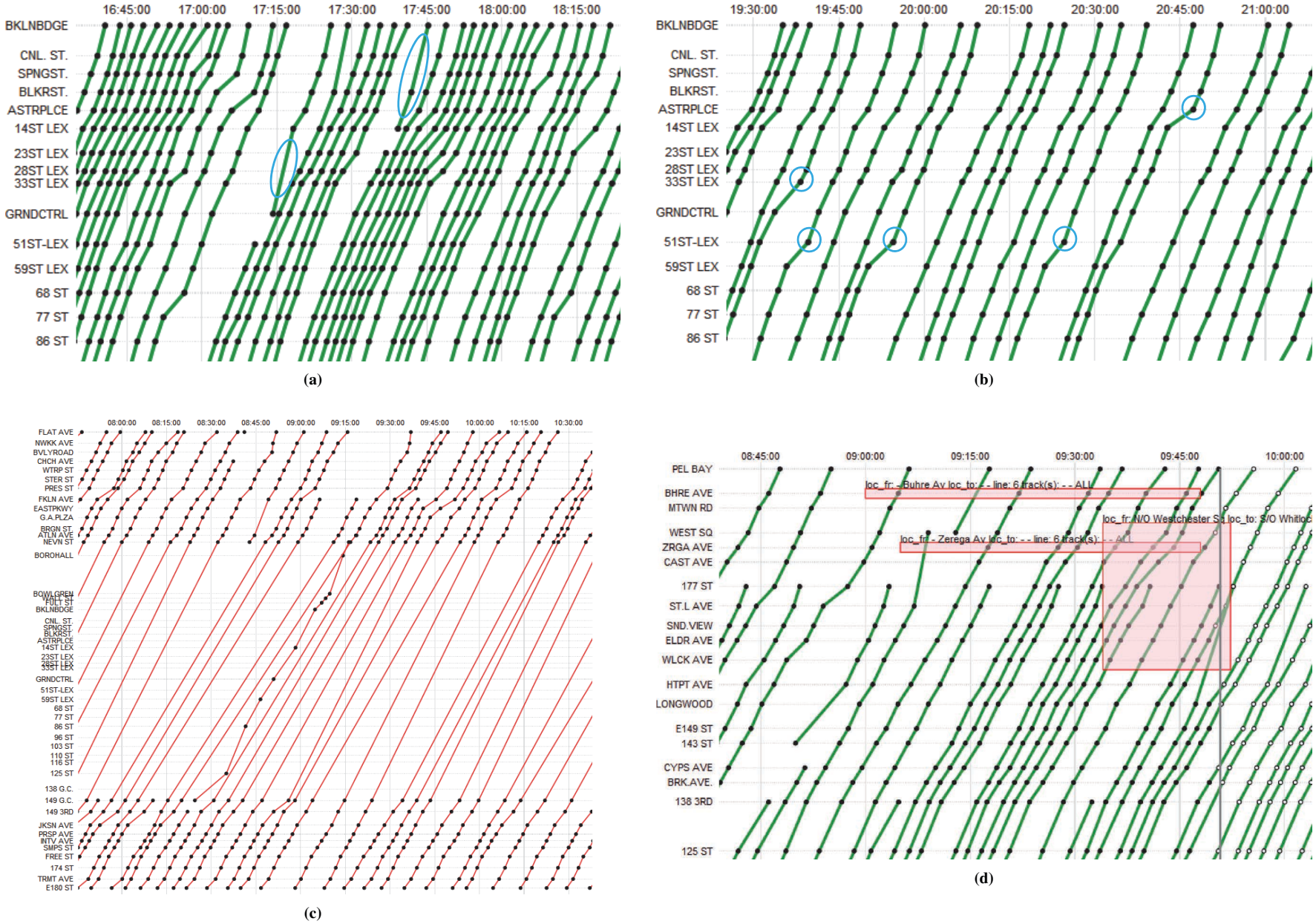

FIGURE 4 Examples of service management actions and flagging visualized on stringlines: (*a*) two trains skipping stops (points are missing from those lines), (*b*) trains held in stations (holds circled in blue), (*c*) Line 2 train rerouted along Line 5 in Manhattan, and (*d*) flagging zones and times shown as rectangles.

EVALUATION AND ANALYSIS

The stringlines tool was evaluated using three criteria: usability and interface design; representativeness of train service by a service scenario analysis; and end-user meetings, training, and testing. Like the development process itself, all three criteria were reviewed multiple times in an iterative process involving numerous discussions and informal meetings over the course of at least a year. Building relationships with the Department of Subways and RCC staff to determine needs and become familiar with the organizational culture was at least as important as the technological aspects of the project.

Usability and Interface Design

Consistent with the iterative design concept, the user interface was designed with simplicity in mind, and there was no intention of covering every possible future need of the hypothetical user. A choice was made to follow a basic web design paradigm of the main window and the sidebar that controls what is presented in the main window. The functionality was documented in a help file and training materials; when working through the training process, it became clear that most users are comfortable with the paradigm once they understand the different options presented to them in the sidebar. Another challenge was present in the nature of the data. For example, if a train's ID is changed, it may be assigned a different schedule and appear as a distinct line disconnected from the line representing the initial part of the trip. Such problems stem from the current data scheme; however, the intended audience (console dispatchers) would be the ones actually making the service changes and would therefore see how their actions cause corresponding changes to the stringlines.

Service Scenario Analysis

Some of the earliest uses of the stringlines tool were service analyses requested by managers who wanted to corroborate their personal experiences of riding the subway. This request provided a chance to prove that what the stringlines show is actually representative of real service. The stringlines application presented an opportunity to see service in real time outside of the RCC, which led to its use by Operations Planning managers and staff.

Looking at the data visualization made headway regularity problems readily apparent. For example, a given station and arrival time could be looked up on the stringline chart to provide an explanation for whether a delay was caused by volume (an aggregation of many small delays) or an individual train, likely involved in an incident. Figure 4*a* illustrates gaps in service and bunching and is also an example of two trains instructed to skip stops to make up for the gap ahead of them. Figure 4*b* shows the effect that holding trains can have to even out headways.

The hybrid display paradigm that allows for viewing train stringlines from either the point of view of that line's stations or another line's stations made for easier analysis of service changes, especially reroutes. For instance, in Figure 4*c*, the Line 2 train that has been rerouted to the Lexington Avenue IRT can be picked out immediately because, by switching to the Line 5 stations' view but selecting the Line 2's check box, any Line 2 trains that stopped on the Lexington Avenue line will have data points in Manhattan, matching the service pattern of Line 5 trains.

Training and Testing

Building a closer working relationship with RCC staff, Operations Planning has been able to develop a pilot initiative involving the sharing and analysis of multiple data sources, such as flagging and incident data. In the beginning, meetings were held with management to discuss the stringlines tool conceptually. Later, the stringlines web application was tested on computers set up next to ATS-A consoles. Evaluating the stringline view and the model board view side by side was the first true test for the stringlines tool. It became clear that real-time performance was the critical feature for the system: immediate feedback is a key expectation in RCC work. Being present at the RCC and actively interacting with daily users of ATS-A proved to be very important for both getting the staff to accept the new technology and understanding their needs in order to go back and improve the stringlines tool.

LESSONS LEARNED

The GTFS-RT stringlines project has been a tremendous learning opportunity in the field of rapidly prototyping applications for transit service management. The transformative power of this approach may prove to be useful across the agency as it picks up trends already accepted in the private sector. Combining standards-based software and an iterative development process, it became apparent that integrating this new technology would work best if close contacts are maintained and appropriate training is offered when needed to facilitate culture change and adoption. The following are the main lessons learned from this project:

- Benefits of iterative development. Short cycles of development, testing, and feedback allow applications to be built quickly and with maximum flexibility to modify features. Building relationships with users enabled the development team to customize the front end of the stringlines tool to fit the needs of operations.
- Standards and open-source tools enable in-house development. Working with open-source tools and public data feeds allows for fast development of new applications with minimal marginal cost. The approach with GTFS-RT data specifically can be used by other properties as development of GTFS-RT feeds for public use continues to become more widespread.
- Performance is critical in real-time data processing. The first version of the stringlines web application was a breakthrough, but its 10- to 15-min time lag made it unsuitable for use in the RCC because it did not match the real-time model board display. This problem motivated streamlining the back-end processing to let it keep up with the 30-s message interval.
- Stringline charts are an effective tool for visualizing service. It is easier to visualize spacing between trains on stringline charts than on the RCC model board display, where distances are not to scale. This enabled managers to conduct service scenario analyses with a focus on wait assessment, which is a measure of headway regularity and NYCT's primary subway service performance indicator.
- Data-driven culture change takes time. RCC dispatchers work in a fast-paced environment, and they are most familiar with monitoring service on the model board. It was important to consider their feedback instead of just providing them with one more screen to look at. This issue led to the development of an additional prototype tool that, using the same back-end process, delivers the most relevant

service headway information as quickly as possible, listing the largest gaps between trains in tabular form.

- Internal data also need to be incorporated. The main limitation of NYCT's GTFS-RT data feed for internal use is that it does not include information on schedule supplements, which accommodate planned track work or station closures. Information on supplements needs to be integrated to correctly evaluate performance indicators in real time.

FUTURE WORK

The stringlines project is a part of a growing suite of tools that are being tested at NYCT and incorporated into a framework that can ultimately help improve service delivery by providing service feedback and related system data. The next steps for the stringlines application are to incorporate internal data sources, including schedule information and data on incidents and track work, to compute performance indicators and provide a more complete picture of conditions in the system. The application will also be integrated into the web reporting tool for improved accessibility.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the contributions of Tuan Huynh and Liu Liu to the development of the GTFS-RT processing algorithm.

REFERENCES

1. Cramer, A., J. Cucarese, M. Tran, A. Lu, and A. Reddy. Performance Measurements on Mass Transit: Case Study of New York City Transit. In *Transportation Research Record: Journal of the Transportation Research Board, No. 2111,* Transportation Research Board of the National Academies, Washington, D.C., 2009, pp. 125–138.
2. Levine, B., A. Lu, and A. V. Reddy. Measurement of Subway Service Performance at New York City Transit: Case Study with Automated Train Supervision Track-Occupancy Data. In *Transportation Research Record: Journal of the Transportation Research Board, No. 2353,* Transportation Research Board of the National Academies, Washington, D.C., 2013, pp. 57–68.
3. Levine, B., S. Iyer, and A. Reddy. Development of an Automated Vehicle Location (AVL) Data Based System to Improve Bus Service at New York City Transit. Presented at 93rd Annual Meeting of the Transportation Research Board, Washington, D.C., 2014.
4. Mann, T. Google Maps Starts Including Real-Time Subway Departure Data. *Wall Street Journal* blog, March 27, 2013. http://blogs.wsj.com/metropolis/2013/03/27/google-maps-starts-including-real-time-subway-departure-data. Accessed March 13, 2015.
5. Tufte, E. R. *The Visual Display of Quantitative Information,* 2nd ed. Graphics Press, Cheshire, Conn., 2001.
6. Frumin, M. *Seeing the Forest and the Trees: Operations Data and Stringline Charts for Improving Service and Schedules in the Department of Subways of MTA New York City Transit.* New York City Transit, New York.
7. Frumin, M., J. Zhao, N. Wilson, and Z. Zhao. Automatic Data for Applied Railway Management: A Case Study on the London Overground. In *Transportation Research Record: Journal of the Transportation Research Board, No. 2353,* Transportation Research Board of the National Academies, Washington, D.C., 2013, pp. 47–56.
8. Bertini, R. L., and A. El-Geneidy. Generating Transit Performance Measures with Archived Data. In *Transportation Research Record: Journal of the Transportation Research Board, No. 1841,* Transportation Research Board of the National Academies, Washington, D.C., 2003, pp. 109–119.
9. *What Is GTFS?* https://developers.google.com/transit/gtfs. Accessed March 13, 2015.
10. Wong, J. Leveraging the General Transit Feed Specification for Efficient Transit Analysis. In *Transportation Research Record: Journal of the Transportation Research Board, No. 2338,* Transportation Research Board of the National Academies, Washington, D.C., 2013, pp. 11–19.
11. Puchalsky, C., D. Joshi, and W. Scherr. Development of a Regional Forecasting Model Based on Google Transit Feed. Presented at 91st Annual Meeting of the Transportation Research Board, Washington, D.C., 2012.
12. Lee, S. G., M. Hickman, and D. Tong. Stop Aggregation Model: Development and Application. In *Transportation Research Record: Journal of the Transportation Research Board, No. 2276,* Transportation Research Board of the National Academies, Washington, D.C., 2012, pp. 38–47.
13. Nazem, M., M. Trépanier, and C. Morency. Integrated Intervening Opportunities Model for Public Transit Trip Generation-Distribution: A Supply-Dependent Approach. In *Transportation Research Record: Journal of the Transportation Research Board, No. 2350,* Transportation Research Board of the National Academies, Washington, D.C., 2013, pp. 47–57.
14. *BART GTFS-Real Time Developer Resources.* http://www.bart.gov/schedules/developers/gtfs-realtime. Accessed March 13, 2015.
15. *MBTA-Realtime Developer Portal.* http://realtime.mbta.com/portal. Accessed March 13, 2015.
16. *TriMet GTFS-Realtime Feeds.* http://developer.trimet.org/GTFS.shtml. Accessed March 13, 2015.
17. New York City Transit. *MTA Releases Real-Time Subway Arrival Times.* Press release, Dec. 28, 2012. http://www.mta.info/press-release/mta-headquarters/mta-releases-real-time-subway-arrival-times. Accessed March 13, 2015.
18. *GTFS-Realtime Reference for the New York City Subway.* http://datamine.mta.info/sites/all/files/pdfs/GTFS-Realtime-NYC-Subway%20version%201%20dated%207%20Sep.pdf. Accessed March 13, 2015.
19. *Time Complexity of Various Operations in Python.* https://wiki.python.org/moin/TimeComplexity. Accessed March 13, 2015.
20. *Data-Driven Documents.* http://d3js.org. Accessed March 13, 2015.
21. *Marey's Trains Drawn Using the D3.js JavaScript Library.* http://bl.ocks.org/mbostock/5544008. Accessed March 13, 2015.
22. New York City Transit. *MTA Adds Real-Time Arrival Estimates on L Line to Subway Time App, Website and Open Data Portal.* Press release, Jan. 13, 2014. http://www.mta.info/press-release/nyc-transit/mta-adds-real-time-arrival-estimates-l-line-subway-time-app-website-and. Accessed March 13, 2015.

All errors or omissions are the responsibility of the authors. Opinions expressed are the authors' and do not necessarily reflect official policy or positions of New York City Transit or the Metropolitan Transportation Authority of the State of New York.

The Public Transportation Group peer-reviewed this paper.